SEHH2239 Data Structures
Revision 2

Question 1

B1

(a) Draw the expression tree of the following infix expression. (3 marks)

$$(1 + 2) * (8 – 3) / (5 + 7)$$

(b) Give prefix and postfix forms of the expression. (4 marks)

(c) What are the differences between an expression tree and a binary tree? (3 marks)

Question 2
C1

(a) Figure 1 shows the linked list structure in which the object reference `head` points at the first node and object reference `pp` points at `head.next`.

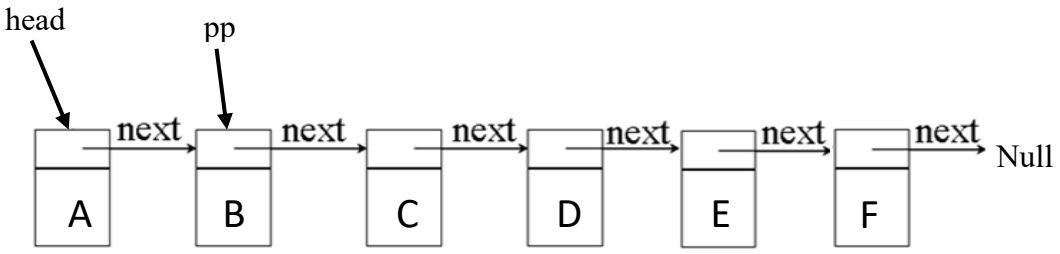

Figure 1

(i) After executing the following Python statements,

```
head.next = head.next.next
pp.next = head.next.next
head.next.next.next = pp.next.next
head.next.next = None
```

the linked list in the above will be changed. Draw all original nodes with the new links. (4 marks)

(ii) Which node will be collected by garbage collector? (1 mark)

(iii) Starting from Figure 1, there is one more object reference `nm` defined. Complete the following Python statements such that after executing these statements, the linked list in Figure 5 will be changed into the two lists shown in Figure 2 and the object reference `head` and `pp` will point to the first node of each list. (Note: Modifying any of the following given parts of the Python statements or adding Python statements are **NOT** allowed.)

(5 marks)

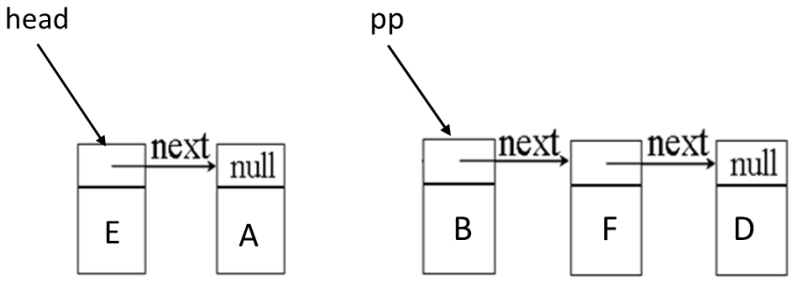

Figure 2

```
nm = head
pp.next.next.next.next.next =
head =
pp.next =
head.next =
head.next.next =
pp.next.next.next =
```

Question 3
C2

```
class Node :
    def __init__(self, element, next=None) :
        self.element = element
        self.next = next


# *** Linked Queue *** #
class LinkedQueue :
    def __init__(self) :
        self.front = None

    # return true if queue is empty
    def isEmpty(self):
        return self.front == None

    def getFrontElement(self) :
        if  self.isEmpty() :
            return None
             # return None if the queue is empty
        else :
            return self.front.element
             # return the element at the queue front

    def getRearElement(self) :
        # Get the last element in queue
        # Your code in (a) should be inserted here


    def put(self, theElement) :
        n = Node(theElement);
        p = None
        if self.front == None :
            self.front = n   # insert the Element into empty queue
        else :
            p = self.front
            while p.next != None :
                p = p.next
            p.next = n      # insert the Element at the queue rear

    def remove(self) :
        # remove an element from the front of the queue
        # Your code in (b) should be inserted here
```

```
if __name__ == "__main__" :
    q = LinkedQueue()
    q.put(int(15))
    q.put(int(18))
    q.put(int(23))
    q.put(int(25))
    x = q.remove()
    q.put(int(36))
    q.put(int(78))
    while q.isEmpty() == False :
        print(  "Front=" + str(q.getFrontElement()) + ", " + \
                "Rear="  + str(q.getRearElement()) + ", "  + \
                "Removed Element=" + str(q.remove()) )
```

(a) Complete the `getRearElement()` method in the *LinkedQueue* class such that the method returns none if the queue is empty, otherwise returns the element at the rear of the queue.

(5 marks)

(b) Complete the `remove()` method in the *LinkedQueue* class such that the method returns none if the queue is empty, otherwise removes an element from the front of the queue and returns the removed element. (5 marks)

(c) Show the output after the given codes are successfully executed. (5 marks)

(d) Draw the **stack** data structures in array implementations for "each step" in the following sequence:

add(A), add(B), remove, remove, add(D), add(E), remove, add(F), add(G).

Assume an initial size of 5 for the array implementation. Remember to show **TOP** (top of the stack) for stack. (Draw the diagram in the answer book.)

(5 marks)

Step